UNITED STATES PATENT APPLICATION

FOR

DISTRIBUTING PACKETS AND PACKETS FRAGMENTS POSSIBLY RECEIVED OUT OF SEQUENCE INTO AN EXPANDABLE SET OF QUEUES OF PARTICULAR USE IN PACKET RESEQUENCING AND REASSEMBLY

INVENTORS:

EARL T. COHEN OF FREMONT, CA

JOHN ANDREW FINGERHUT OF CAMPBELL, CA

JOHN J. WILLIAMS, JR. OF PLEASANTON, CA

ASSIGNEE:
CISCO TECHNOLOGY, INC.
170 W. TASMAN DRIVE, SAN JOSE, CA 95134, A CALIFORNIA CORPORATION

PREPARED BY:
THE LAW OFFICE OF KIRK D. WILLIAMS
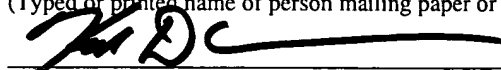1234 S. OGDEN ST., DENVER, CO 80210
303-282-0151

DISTRIBUTING PACKETS AND PACKETS FRAGMENTS POSSIBLY RECEIVED OUT OF SEQUENCE INTO AN EXPANDABLE SET OF QUEUES OF PARTICULAR USE IN PACKET RESEQUENCING AND REASSEMBLY

5

TECHNICAL FIELD

One embodiment of the invention relates to communications and computer systems, especially networked routers, packet switching systems, and other devices; and more particularly, one embodiment relates to distributing packets and/or packets

10   fragments possibly received out of sequence into an expandable set of queues of particular use in packet resequencing and/or reassembly.

15   **BACKGROUND**

The communications industry is rapidly changing to adjust to emerging technologies and ever increasing customer demand. This customer demand for new applications and increased performance of existing applications is driving communications network and system providers to employ networks and systems having

20   greater speed and capacity (e.g., greater bandwidth). In trying to achieve these goals, a common approach taken by many communications providers is to use packet switching technology. Increasingly, public and private communications networks are being built and expanded using various packet technologies, such as Internet Protocol (IP). Note, nothing described or referenced in this document is admitted as prior art to this application unless

25   explicitly so stated.

A network device, such as a switch or router, typically receives, processes, and forwards or discards a packet. Each packet is typically part of a larger set of information being communicated between two entities, and it is desirous and often a requirement that packets of a sequence be delivered in the order specified by that sequence. However,

30   various transmission, packet processing, and packet switching techniques may result in

packets arriving out of order at intermediate or final destinations where the packets are needed in their original order. Thus, a packet resequencing technique must be employed, such as storing the packets in random access memory and removing them from the memory in order.

5   Similarly, a single packet can be split into multiple packet fragments for transmission, processing or other purposes. For example, Multilink Point-to-Point Protocol (MLPPP) and Frame Relay Forum .12 (FRF .12) break a packet into fragments and transmit these multiple packet fragments over multiple physical links or facilities. These packets fragments are each identified with a packet sequence number to identify

10   the order of the fragments. The receiving device must then reassemble the original packets in order from the received packet fragments as well as detect and recover from any lost packet fragments. One feature that makes this packet reassembly and error detection easier is that packets received over one of the links will, in the most prevalent uses of reassembly, also include a monotonically increasing link sequence number for the

15   corresponding link over which the packet fragment is received or for the set of links over which the packet fragments are received. Although these packet fragments may often be received in order, there are situations for which some of these packet fragments may be received out of order.

Known systems performing packet reassembly use random access memory to

20   reassemble the packets. For example, one technique writes each of the received packet fragments in memory at any location and keeps track of the received fragments and their location in memory in a "scorecard" data structure (e.g., it indicates the memory locations in reassembly order). Based on the scorecard, the packet fragments can be read out of memory in order to reassemble the original packet.

25   Another known technique allocates a portion of memory large enough to handle a largest size packet, and then stores each packet fragment in this portion of memory at a position relative to the sequence number of the fragment. In this manner, a packet is

reassembled in memory, and the contiguous memory locations can be read to generate the reassembled packet.

In another context, merge sorting is a known technique for producing a single sorted list from multiple ordered lists whose values are known a priori. For example, two lists of known elements sorted in ascending order can be combined into a single sorted list by repeatedly taking the smaller value from the front of lists, and appending the smaller value to the end of the combined list. This example can be extended to a set of n known values, which can be sorted by first dividing the set into n lists containing a single value each, then combining pairs of lists to produce $n/2$ lists with two values each. Pairs of these lists are then merged, producing $n/4$ lists with four values each. Continuing in this fashion eventually yields a single sorted list containing the original values, but in sorted order. Merge sorting can also be implemented using three-way merging (that is, merging three sorted lists into a single sorted list in one step), rather than by using two-way merging. More generally, d-way merging can be used for any integer $d > 1$. However, merge sorting cannot be used to produce a merged list of items by merging lists where one or more of the lists is not already in sorted order, as the merging technique can only take an item from the front of an input list, and not a next item in order that is not at the front of one of the input lists.

65609

SUMMARY

Disclosed are, *inter alia*, methods, apparatus, data structures, computer-readable media, mechanisms, and means for distributing packets and/or packets fragments possibly received out of sequence into an expandable set of queues of particular use in packet

5   resequencing and/or reassembly. Typically for each particular packet or fragment, a queue within a set of queues is identified that does not contain a packet or packet fragment that is subsequent to the particular packet or fragment, and the particular packet or fragment is enqueued therein. If there is no such queue available, a new queue is added to the set of queues. Note, a fragment can be enqueued in a queue that contains a fragment whose

10   position is earlier in the reassembled packet to which the fragments belong.

By storing fragments in this arrangement, first-in first-out (FIFO) queues can be used and fragments can be removed from the queues in the order they belong in the reassembled packet. FIFO queues may be implemented very efficiently in hardware and at rates faster than that available using random access memory. (FIFO queues eliminate the

15   need to track the addresses of specific packet fragments or packets, which makes it more efficient in hardware to support an arbitrarily large number of packet fragments or packets.)

A data structure is typically updated for packet fragments after the queue is determined to reflect when all fragments of a given packet have been received. The data

20   structure specifies the order of queues containing the packet fragments in order of their position within the reassembled packet. This ordered list of the queues is communicated to a reassembly mechanism to retrieve the packet fragments and to reassemble the packet. Resequencing of packets is similarly performed and may be part of the reassembly process. In one embodiment, the list of queues is not used by the reassembly/resequencing

25   mechanism as the fragments/packets stored in the queues contain sequence numbers, and thus their ordering can identified from the queued fragments/packets.

One embodiment reassembles a packet. A set of multiple queues is maintained. For each particular packet fragment, a particular queue of the set of queues is identified

4

such that no already queued packet fragment in the particular queue comes after the particular packet fragment in the reassembled packet. This identification includes checking, at least when the particular queue is not empty, that the particular queue does not include a packet fragment which comes after the particular packet fragment. The

5   particular packet fragment is then enqueued at the end of the particular queue, and a data structure is updated to include an indication of the particular queue in relation to the position of the particular packet fragment in the packet. The packet fragments are dequeued from the packet queues based on the data structure to produce the reassembled packet, such as in response to identifying that all of packet fragments have been

10   enqueued.

       In one embodiment, in response to identifying that no queue is available to enqueue the particular packet fragment (i.e., each queue has a fragment which comes after the particular packet fragment in the reassembled packet), the set of queues is expanded by adding the particular queue to the set of queues. One embodiment reduces the number

15   of queues in the set of queues in response to identifying that the particular queue is empty, which may include removing the particular queue from the set of queues. In one embodiment, the packet fragments are retrieved from the set of queues in order of their respective position within the packet. In one embodiment, the data structure corresponds to a meta-packet or scorecard including the indications of the queues in an order

20   corresponding to the positions of their corresponding packet fragments within the reassembled packet. In one embodiment, at least two packet fragments of the packet are enqueued in a same queue.

       One embodiment resequences packets. A set of queues is maintained. A particular queue is identified for each packet of a stream of packets based on the packet's sequence

25   number such that no packet subsequent to the packet has already been enqueued in the queue, and the corresponding packet is enqueued therein. Packets can then be retrieved in their order from the set of queues to form the resequenced stream of packets. In one embodiment, if no such queue is currently available, a queue is added to the set of queue,

and the packet enqueued therein. One embodiment, reduces the number of queues in response to identifying that one of the queues is empty (e.g., an empty queue is removed from the set of queues).

One embodiment resequences and reassembles packets. Multiple queues are

5   maintained. A packet fragment of multiple packet fragments of a packet of a stream of packets is received. A particular queue is identified by checking a data structure or the queues for the particular packet fragment such that no already queued packet fragment in the particular queue comes after the particular packet fragment in the packet and the particular queue includes no already queued packet fragment of a packet subsequent to

10   the particular packet in the stream of packets. The particular packet fragment is enqueued at the end of the particular queue, and a data structure is updated to include an indication of the particular queue in relation to the position of the particular packet fragment in the packet and the stream of packets. The packet fragments are retrieved based on the data structure to produce the packet. In one embodiment, the packet fragments for a packet

15   are retrieved in response to identifying that all of the plurality of packet fragments have been enqueued in the plurality of queues and no other packet prior to the packet in the stream of packets is enqueued in the plurality of queues. In one embodiment, the number of queues available for enqueuing packet fragments can dynamically increase and decrease.

20

BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the invention with particularity. The invention, together with its advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

5   FIG. 1A is a block diagram of a reassembly and/or resequencing system used in one embodiment;

FIG. 1B is a block diagram of a system or component used in one embodiment for distributing packets and/or packet fragments to a set of queues and/or for packet resequencing and/or reassembly;

10   FIG. 2 is a block diagram illustrating different data structures used in one embodiment;

FIG. 3A is a flow diagram of a process for distributing packet fragments to a set of queues used in one embodiment;

FIG. 3B is a flow diagram of a process for removing packet fragments from a set

15   of queues and reassembling packets used in one embodiment;

FIG. 4A is a flow diagram of a process for distributing packets to a set of queues used in one embodiment; and

FIG. 4B is a flow diagram of a process for removing packets from a set of queues in resequencing a stream packets used in one embodiment.

20

DETAILED DESCRIPTION

Disclosed are, *inter alia*, methods, apparatus, data structures, computer-readable media, mechanisms, and means for distributing packets and/or packets fragments possibly received out of sequence into an expandable set of queues of particular use in packet

5   resequencing and/or reassembly.

Embodiments described herein include various elements and limitations, with no one element or limitation contemplated as being a critical element or limitation. Each of the claims individually recites an aspect of the invention in its entirety. Moreover, some embodiments described may include, but are not limited to, inter alia, systems, networks,

10   integrated circuit chips, embedded processors, ASICs, methods, and computer-readable media containing instructions. One or multiple systems, devices, components, etc. may comprise one or more embodiments, which may include some elements or limitations of a claim being performed by the same or different systems, devices, components, etc. The embodiments described hereinafter embody various aspects and configurations within the

15   scope and spirit of the invention, with the figures illustrating exemplary and non-limiting configurations. Note, computer-readable media and means for performing methods and processing block operations are disclosed and are in keeping with the extensible scope and spirit of the invention.

As used herein, the term "packet" refers to packets of all types or any other units

20   of information or data, including, but not limited to, fixed length cells and variable length packets, each of which may or may not be divisible into smaller packets or cells. The term "packet" as used herein also refers to both the packet itself or a packet indication, such as, but not limited to all or part of a packet or packet header, a data structure value, pointer or index, or any other part or direct or indirect identification of a packet or information

25   associated therewith. For example, often times a router operates on one or more fields of a packet, especially the header, so the body of the packet is often stored in a separate memory while the packet header is manipulated, and based on the results of the processing of the packet (i.e., the packet header in this example), the entire packet is

forwarded or dropped, etc. Additionally, these packets may contain one or more types of information, including, but not limited to, voice, data, video, and audio information. The term "item" is used generically herein to refer to a packet or any other unit or piece of information or data, a device, component, element, or any other entity. The phrases

5   "processing a packet" and "packet processing" typically refer to performing some steps or actions based on the packet contents (e.g., packet header or other fields), and such steps or action may or may not include modifying, storing, dropping, and/or forwarding the packet and/or associated data.

The term "system" is used generically herein to describe any number of

10   components, elements, sub-systems, devices, packet switch elements, packet switches, routers, networks, computer and/or communication devices or mechanisms, or combinations of components thereof. The term "computer" is used generically herein to describe any number of computers, including, but not limited to personal computers, embedded processing elements and systems, control logic, ASICs, chips, workstations,

15   mainframes, etc. The term "processing element" is used generically herein to describe any type of processing mechanism or device, such as a processor, ASIC, field programmable gate array, computer, etc. The term "device" is used generically herein to describe any type of mechanism, including a computer or system or component thereof. The terms "task" and "process" are used generically herein to describe any type of running program,

20   including, but not limited to a computer process, task, thread, executing application, operating system, user process, device driver, native code, machine or other language, etc., and can be interactive and/or non-interactive, executing locally and/or remotely, executing in foreground and/or background, executing in the user and/or operating system address spaces, a routine of a library and/or standalone application, and is not limited to

25   any particular memory partitioning technique. The steps, connections, and processing of signals and information illustrated in the figures, including, but not limited to any block and flow diagrams and message sequence charts, may typically be performed in the same or in a different serial or parallel ordering and/or by different components and/or

processes, threads, etc., and/or over different connections and be combined with other functions in other embodiments, unless this disables the embodiment or a sequence is explicitly or implicitly required (e.g., for a sequence of read the value, process the value - the value must be obtained prior to processing it, although some of the

5   associated processing may be performed prior to, concurrently with, and/or after the read operation). Furthermore, the term "identify" is used generically to describe any manner or mechanism for directly or indirectly ascertaining something, which may include, but is not limited to receiving, retrieving from memory, determining, defining, calculating, generating, etc.

10   Moreover, the terms "network" and "communications mechanism" are used generically herein to describe one or more networks, communications media or communications systems, including, but not limited to the Internet, private or public telephone, cellular, wireless, satellite, cable, local area, metropolitan area and/or wide area networks, a cable, electrical connection, bus, etc., and internal communications

15   mechanisms such as message passing, interprocess communications, shared memory, etc. The term "message" is used generically herein to describe a piece of information which may or may not be, but is typically communicated via one or more communication mechanisms of any type.

     The term "storage mechanism" includes any type of memory, storage device or

20   other mechanism for maintaining instructions or data in any format. "Computer-readable medium" is an extensible term including any memory, storage device, storage mechanism, and other storage and signaling mechanisms including interfaces and devices such as network interface cards and buffers therein, as well as any communications devices and signals received and transmitted, and other current and evolving technologies

25   that a computerized system can interpret, receive, and/or transmit. The term "memory" includes any random access memory (RAM), read only memory (ROM), flash memory, integrated circuits, and/or other memory components or elements. The term "storage device" includes any solid state storage media, disk drives, diskettes, networked services,

tape drives, and other storage devices. Memories and storage devices may store computer-executable instructions to be executed by a processing element and/or control logic, and data which is manipulated by a processing element and/or control logic. The term "data structure" is an extensible term referring to any data element, variable, data

5   structure, database, and/or one or more organizational schemes that can be applied to data to facilitate interpreting the data or performing operations on it, such as, but not limited to memory locations or devices, sets, queues, trees, heaps, lists, linked lists, arrays, tables, pointers, etc. A data structure is typically maintained in a storage mechanism. The terms "pointer" and "link" are used generically herein to identify some mechanism for

10   referencing or identifying another element, component, or other entity, and these may include, but are not limited to a reference to a memory or other storage mechanism or location therein, an index in a data structure, a value, etc.

      The term "one embodiment" is used herein to reference a particular embodiment, wherein each reference to "one embodiment" may refer to a different embodiment, and

15   the use of the term repeatedly herein in describing associated features, elements and/or limitations does not establish a cumulative set of associated features, elements and/or limitations that each and every embodiment must include, although an embodiment typically may include all these features, elements and/or limitations. In addition, the phrase "means for xxx" typically includes computer-readable medium containing

20   computer-executable instructions for performing xxx.

      In addition, the terms "first," "second," etc. are typically used herein to denote different units (e.g., a first element, a second element). The use of these terms herein does not necessarily connote an ordering such as one unit or event occurring or coming before another, but rather provides a mechanism to distinguish between particular units.

25   Additionally, the use of a singular tense of a noun is non-limiting, with its use typically including one or more of the particular thing rather than just one (e.g., the use of the word "memory" typically refers to one or more memories without having to specify "memory or memories," or "one or more memories" or "at least one memory", etc.). Moreover, the

phrases "based on x" and "in response to x" are used to indicate a minimum set of items x from which something is derived or caused, wherein "x" is extensible and does not necessarily describe a complete list of items on which the operation is performed, etc. Additionally, the phrase "coupled to" is used to indicate some level of direct or indirect

5   connection between two elements or devices, with the coupling device or devices modifying or not modifying the coupled signal or communicated information. The term "subset" is used to indicate a group of all or less than all of the elements of a set. The term "subtree" is used to indicate all or less than all of a tree. Moreover, the term "or" is used herein to identify a selection of one or more, including all, of the conjunctive items.

10   Disclosed are, *inter alia*, methods, apparatus, data structures, computer-readable media, mechanisms, and means for distributing packets and/or packets fragments possibly received out of sequence into an expandable set of queues of particular use in packet resequencing and/or reassembly. Typically for each particular packet or fragment, a queue within a set of queues is identified that does not contain a packet or packet fragment that

15   is subsequent to the particular packet or fragment, and the particular packet or fragment is enqueued therein. If there is no such queue available, a new queue is added to the set of queues. Note, a fragment can be enqueued in a queue that contains a fragment whose position is earlier in the reassembled packet to which the fragments belong.

By storing fragments in this arrangement, first-in first-out (FIFO) queues can be

20   used and fragments can be removed from the queues in the order they belong in the reassembled packet. FIFO queues may be implemented very efficiently in hardware and at rates faster than that available using random access memory. (FIFO queues eliminate the need to track the addresses of specific packet fragments or packets, which makes it more efficient in hardware to support an arbitrarily large number of packet fragments or

25   packets.)

A data structure is typically updated for packet fragments after the queue is determined to reflect when all fragments of a given packet have been received. The data structure specifies the order of queues containing the packet fragments in order of their

position within the reassembled packet. This ordered list of the queues is communicated to a reassembly mechanism to retrieve the packet fragments and to reassemble the packet. Resequencing of packets is similarly performed and may be part of the reassembly process. In one embodiment, the list of queues is not used by the reassembly/resequencing mechanism as the fragments/packets stored in the queues contain sequence numbers, and thus their ordering can identified from the queued fragments/packets.

One embodiment reassembles a packet. A set of multiple queues is maintained. For each particular packet fragment, a particular queue of the set of queues is identified such that no already queued packet fragment in the particular queue comes after the particular packet fragment in the reassembled packet. This identification includes checking, at least when the particular queue is not empty, that the particular queue does not include a packet fragment which comes after the particular packet fragment. The particular packet fragment is then enqueued at the end of the particular queue, and a data structure is updated to include an indication of the particular queue in relation to the position of the particular packet fragment in the packet. In response to identifying that all of packet fragments have been enqueued, the packet fragments are dequeued from the packet queues based on the data structure to produce the reassembled packet.

In one embodiment, in response to identifying that no queue is available to enqueue the particular packet fragment (i.e., each queue has a fragment which comes after the particular packet fragment in the reassembled packet), the set of queues is expanded by adding the particular queue to the set of queues. One embodiment reduces the number of queues in the set of queues in response to identifying that the particular queue is empty, which may include removing the particular queue from the set of queues. In one embodiment, the packet fragments are retrieved from the set of queues in order of their respective position within the packet. In one embodiment, the data structure corresponds to a meta-packet or scorecard including the indications of the queues in an order corresponding to the positions of their corresponding packet fragments within the

13

reassembled packet. In one embodiment, at least two packet fragments of the packet are enqueued in a same queue.

One embodiment resequences packets. A set of queues is maintained. A particular queue is identified for each packet of a stream of packets based on the packet's sequence number such that no packet subsequent to the packet has already been enqueued in the queue, and the corresponding packet is enqueued therein. Packets can then be retrieved in their order from the set of queues to form the resequenced stream of packets. In one embodiment, if no such queue is currently available, a queue is added to the set of queue, and the packet enqueued therein. One embodiment, reduces the number of queues in response to identifying that one of the queues is empty (e.g., an empty queue is removed from the set of queues).

One embodiment resequences and reassembles packets. Multiple queues are maintained. A packet fragment of multiple packet fragments of a packet of a stream of packets is received. A particular queue is identified by checking a data structure or the queues for the particular packet fragment such that no already queued packet fragment in the particular queue comes after the particular packet fragment in the packet and the particular queue includes no already queued packet fragment of a packet subsequent to the particular packet in the stream of packets. The particular packet fragment is enqueued at the end of the particular queue, and a data structure is updated to include an indication of the particular queue in relation to the position of the particular packet fragment in the packet and the stream of packets. The packet fragments are retrieved based on the data structure to produce the packet. In one embodiment, the packet fragments for a packet are retrieved in response to identifying that all of the plurality of packet fragments have been enqueued in the plurality of queues and no other packet prior to the packet in the stream of packets is enqueued in the plurality of queues. In one embodiment, the number of queues available for enqueuing packet fragments can dynamically increase and decrease.

In one embodiment, the packet fragments themselves have sequence numbers, and arrive over a set of physical links with the property that the sequence numbers are monotonically increasing on a given physical link (as in MLPPP). By assigning FIFO queues per physical link, and by using a data structure recording which FIFO queue

5   contains each arrived sequence number, the packet fragments comprising a given packet can be reassembled, and the packets can be resequenced, by reading the appropriate packet fragments from the heads of the FIFO queues (as indicated by the data structure), once the last packet fragment of a packet has arrived.

In one embodiment, the arrival of a sequence number on a physical link completes

10   the data structure to a point where it is known that a packet fragment has been lost (or a rule such as sequence numbers monotonically increasing on a given physical link would be violated). The data structure can then be used to drop all the packet fragments up to and including the missing packet fragment, plus any other fragments required to get to the start of the next packet.

15   One embodiment resequences packets. A set of queues is maintained. A particular queue is identified for each packet of a stream of packets based on the packet's sequence number such that no packet subsequent to the packet has already been enqueued in the queue, and the corresponding packet is enqueued therein. Packets can then be retrieved in their order from the set of queues to form the resequenced stream of packets. In one

20   embodiment, if no such queue is currently available, a queue is added to the set of queue, and the packet enqueued therein. One embodiment, reduces the number of queues in response to identifying that one of the queues is empty (e.g., an empty queue is removed from the set of queues).

In one embodiment, the data structure or scorecard is directly indexed by the

25   sequence number of the arriving packet fragment. As each packet fragment arrives, the FIFO queue number on which it is enqueued is written in to the data structure or scorecard at an index determined directly by its sequence number. When the last packet fragment of a packet arrives, a consecutive group of entries in the data structure or

scorecard will be valid, indicating that the corresponding packet is available for reassembly. Furthermore, assuming that no packets are lost, a packet fragment will arrive which completes such a consecutive group of entries at the head of the data structure or scorecard; this group can then be used to provide reassembly directions for the

5   corresponding packet or packets at the heads of the FIFO queues, and the consecutive group of entries can then be removed from the data structure or scorecard.

One embodiment maintains the FIFO queues and operates the reassembly/resequencing mechanism with hardware processes, whereas the data structure or scorecard is maintained and updated by software processes, such as by code running on

10   a network processor.

FIG. 1A is a block diagram of a reassembly and/or resequencing system 100 used in one embodiment. As shown, incoming packets/fragments 101 are typically stored in one or more temporary storage buffers 104. For example, packet fragments of a packet could be received over multiple links and temporarily stored in buffers 104 for processing

15   by distributor 106. Distributor 106 processes the acquired packets/fragments by distributing packets/fragments 108 to a set of queues 110, with the active set of queues typically being expandable/shrinkable to accommodate the distribution requirements. For example, more active queues are required when packets/fragments are received out of order, and especially when smaller sequence-numbered packets/fragments are received

20   much later than larger sequence-numbered packets/fragments. Set of queues 110 typically includes a set of active queues which are currently in use, as well as a pool of expansion queues which can be added to and removed from the active set of queues as needed. These expansion queues are possibly used for other purposes when they are not in the active set of queues. The distributed packets/fragments 108 are placed in queues 110 such

25   that no packet/fragment subsequent to another packet/fragment is placed before it in a same queue. In this manner, packets/fragments can be retrieved in sequence directly from the set of queues 110. Thus, FIFO queues can be employed. FIFO queues can be implemented in hardware such that they are more efficient and faster than that available

using random access memory, which allows packets/fragments to be resequenced/reassembled for lower hardware cost and at very high line rates.

Distributor 106 typically maintains a data structure to identify the order of queues that contain packets/fragments in the order within the sequence of fragments or reassembled packets. This information is typically communicated as indicated by reference number 118 to resequencer/reassembly mechanism 114, such that it can readily remove packets/fragments from set of queues 110 in the desired order as indicated by reference number 112, so as to produce the desired resequenced packet streams / reassembled packets 116.

FIG. 2 illustrates a few data structures used in one embodiment. Queue entry 200, used in one embodiment, includes a sequence number 201, an indication 202 of whether the stored fragment corresponds or does not correspond to a start of packet or end of packet (used in one embodiment that reassembles packets), and data 203 (e.g., the packet or fragment).

Scorecard 210 data structure is used in one embodiment, with scorecard 210 including a list 211-212 of queues corresponding to the resequencing/reassembly order.

Meta-packet 220 data structure similarly provides a list of queues 223-224 corresponding to the resequencing/reassembly order, and typically includes an operation code 221 to identify the type of packet, and optionally a constant value 222 which may be added to the reassembled packet for example.

Ring buffer 250 is used in one embodiment for maintaining indications of received and enqueued packets or packet fragments. Illustrated is a storage for packet 260 with corresponding packet fragment indications 261-262 and for a next packet 270 with corresponding packet fragment indications 271-272. In one embodiment which resequences whole packets, a single entry is included in ring buffer 260 for each packet (e.g., there are no packet fragments and 261-262 correspond to a single queue identifier).

Typically when a first packet fragment of a packet is received, space is allocated in ring buffer 250 for all fragments of the corresponding packet. In one embodiment in

which packet resequencing is also performed, if a packet fragment is received for a packet which comes after another packet in sequence for which space has not already been allocated (e.g., one of its packet fragments has not been previously received), then space is allocated in ring buffer 250 for the previous one or more packets as well as the packet

5   to which the received fragment belongs.

Thus, in one embodiment, a required packet sequence includes packet 260 followed by next packet 270. However, in one embodiment, next packet 270 corresponds to some packet for which a first packet fragment was received after a first packet fragment of packet 260, and the order of packets 260 and 270 do not matter. In fact, in

10   one embodiment, if all fragments of packet 270 are received before all fragments of packet 260 (as determined, for example, by each entry 271-272 including a queue identifier), then packet 270 is allowed to be sent out prior to packet 260. In one embodiment, packet 260 is always sent prior to sending packet 270 (assuming all packet fragments are timely received), whether due a resequencing requirement or the

15   implementation of the embodiment.

FIG. 1B is a block diagram of a system or component 140 used in one embodiment for distributing packets and/or packet fragments to a set of queues 147 and/or for packet resequencing and/or reassembly of packets/fragments stored in queues 147 or elsewhere. In one embodiment, system or component 140 performs one or more

20   processes corresponding to one of the flow diagrams illustrated or otherwise described herein. For example, in one embodiment, distributor and/or reassembly/resequencing mechanism processes run on processing element 141 using queues 147 for reassembling packet fragments and/or resequencing packets.

In one embodiment, system or component 140 includes a processing element 141,

25   memory 142, storage devices 143, an interface 144 for sending and receiving packets/fragments, items, and/or other information, and queues 147, which are typically coupled via one or more communications mechanisms 149, with the communications paths typically tailored to meet the needs of the application. Various embodiments of

component 140 may include more or less elements. The operation of component 140 is typically controlled by processing element 141 using memory 142 and storage devices 143 to perform one or more tasks or processes. Memory 142 is one type of computer-readable medium, and typically comprises random access memory (RAM), read

5   only memory (ROM), flash memory, integrated circuits, and/or other memory components. Memory 142 typically stores computer-executable instructions to be executed by processing element 141 and/or data which is manipulated by processing element 141 for implementing functionality in accordance with an embodiment. Storage devices 143 are another type of computer-readable medium, and typically comprise solid

10   state storage media, disk drives, diskettes, networked services, tape drives, and other storage devices. Storage devices 143 typically store computer-executable instructions to be executed by processing element 141 and/or data which is manipulated by processing element 141 for implementing functionality in accordance with an embodiment. Queues 147 typically correspond to hardware-implemented FIFO queues. However, one

15   embodiment does not include or use queues 147, but rather implements the required set of queues using memory 142.

FIG. 3A is a flow diagram of a process for distributing packet fragments to a set of queues used in one embodiment. Processing begins with process block 300, and proceeds to process block 302, wherein a packet fragment is acquired of a packet to be

20   reassembled and possibly resequenced. A determination is made in process block 304 as to whether a data structure (e.g., a new data structure or space in an existing data structure) has already been established for the corresponding reassembled packet, and possibly for fragments of packets prior in sequence to the packet to which the acquired packet belongs. If not, in process block 306, space is allocated for use in reassembly and

25   possibly resequencing of the packet (e.g., space for all fragments of the packet to which the acquired packet belongs, and possibly for fragments of packets earlier in a sequence of packets). For example, one embodiment uses a ring buffer, table or array to keep track

of received fragments and the queues in which the fragments are enqueued for each active packet being reassembled.

Next, in process block 308, a particular queue is identified which does not have a fragment enqueued to it which is subsequent to the current packet fragment, and if resequencing of packet is being performed, then also does not have a fragment of a packet subsequent to the packet to which the acquired packet fragment belongs. This processing can be performed in an unlimited number of ways, such as maintaining a latest sequence number enqueued per queue for the packet being reassembled, checking the maintained data structure, etc. As determined in process block 310, if no such queue is available, then in process block 312, another queue is added to the set of active queues and will be used to enqueue the current fragment. In process block 314, the current fragment is enqueued to the identified particular queue and an identifier of the identified particular queue is added to the maintained data structure so as to identify the queue and the current fragment's position in the reassembled packet.

As determined in process block 316, if all fragments have been received for a packet being reassembled, then in process block 318, an identification of the queues having a corresponding fragment enqueued therein in the reassembly order is communicated to the reassembly mechanism typically via a meta-packet, scorecard or other communication mechanism, and the corresponding space is freed in the data structure in the distributor. Otherwise, as determined in process block 320, if a timeout has occurred (i.e., the fragments should be purged from the queues and the packet to be reassembled dropped), then in process block 322, an instruction to drop the packet being reassembled and an identification of the queues having a corresponding fragment enqueued therein in the reassembly order is communicated to the reassembly mechanism typically via a meta-packet, scorecard or other communication mechanism (or the queues and the number of fragments in each queue to purge), and the corresponding space is freed in the data structure in the distributor. Processing returns to process block 302 to acquire and process another packet fragment.

FIG. 3B is a flow diagram of a process for removing packet fragments from a set of queues and reassembling packets used in one embodiment. Processing begins with process block 340, and proceeds to process block 342 wherein an instruction is acquired from the distributor to reassemble or drop the packet. As determined in process

5   block 344, if the packet fragments are to be dropped, then in process block 346, the corresponding packet fragments are retrieved from the queues and dropped. Otherwise, in process block 348, the packets are retrieved from the queues, typically in the order specified in the instruction (e.g. meta-packet, scorecard, etc.), and the reassembled packet is forwarded or otherwise processed. As determined in process block 350, if a queue has

10   become empty, then in process block 352, one embodiment frees the queue (i.e., removes it from the active set of queues). Processing returns to process block 342 to acquire and process a next instruction.

FIG. 4A is a flow diagram of a process for distributing packets to a set of queues used in one embodiment. Processing begins with process block 400, and proceeds to

15   process block 402, wherein a packet is acquired for resequencing (e.g., packets delayed too long are ignored). A determination is made in process block 404 as to whether a data structure (e.g., a new data structure or space in an existing data structure) has already been established for the corresponding packet stream being resequenced. If not, in process block 406, space is allocated for use in resequencing of the packet stream (e.g., a ring

20   buffer or other data structure).

Next, in process block 408, a particular queue is identified which does not have a packet enqueued to it which is subsequent to the current packet in the stream of packets. This processing can be performed in an unlimited number of ways, such as maintaining a latest sequence number enqueued per queue for the packet stream being resequenced,

25   checking the maintained data structure, etc. As determined in process block 410, if no such queue is available, then in process block 412, another queue is added to the set of active queues and will be used to enqueue the current packet. In process block 414, the current packet is enqueued to the identified particular queue and an identifier of the

identified particular queue is added to the maintained data structure so as to identify the queue and the current packet's position in the resequenced stream of packets.

As determined in process block 416, if there is a succession of packets in the queues that can be resequenced (e.g., a sequence of one or more packets with sequence
5   numbers beginning with the next sequence number), then in process block 418, an identification of the queues having a corresponding packet enqueued therein in the resequencing order is communicated to the resequencing mechanism typically via a meta-packet, scorecard or other communication mechanism, and the corresponding space is freed in the data structure in the distributor. Otherwise, as determined in process
10   block 420, if a timeout has occurred (i.e., the missing packet should be dropped to allow subsequent packets to proceed), then in process block 422, the corresponding space is freed in the data structure in the distributor for the one or more packets to be dropped, and processing returns to process block 416. Processing then returns to process block 402 to acquire and process another packet.
15   FIG. 4B is a flow diagram of a process for removing packets from a set of queues in resequencing a stream packets used in one embodiment. Processing begins with process block 440, and proceeds to process block 442 wherein an instruction is acquired from the distributor. In process block 444, the packets are retrieved from the queues, typically in the order specified in the instruction (e.g. meta-packet, scorecard, etc.), and
20   the packet or packets are is forwarded or otherwise processed in their resequenced order. As determined in process block 446, if a queue has become empty, then in process block 448, one embodiment frees the queue (i.e., removes it from the active set of queues). Processing returns to process block 442 to acquire and process a next instruction.

In view of the many possible embodiments to which the principles of our
25   invention may be applied, it will be appreciated that the embodiments and aspects thereof described herein with respect to the drawings/figures are only illustrative and should not be taken as limiting the scope of the invention. For example and as would be apparent to one skilled in the art, many of the process block operations can be re-ordered to be

performed before, after, or substantially concurrent with other operations. Also, many different forms of data structures could be used in various embodiments. The invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

5